KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

DISKREČIOSIOS STRUKTŪROS (P170B008)

KURSINIS DARBAS

Užduoties nr. 25 (B lygis)

Atliko:

IFF-8/7 gr. studentas
Gustas Klevinskas

Priėmė:

Dėst. Jūratė Pauliutė

KAUNAS

2019

Turinys

1. Užduotis (nr. 25, B lygis) 3

2. Užduoties analizė 3

3. Programos algoritmo aprašymas 4

4. Programos tekstas 4

5. Testavimo pavyzdžiai 9

   PIRMAS TESTAS ***9***

   ANTRAS TESTAS ***10***

   TREČIAS TESTAS ***11***

6. Išvados 12

7. Literatūros sąrašas 12

1. Užduotis (nr. 25, B lygis)

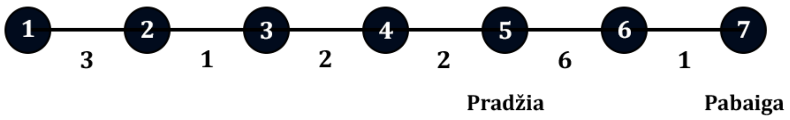
Sudaryti grafą, kur viršūnės atitinka plokštumos taškus, briaunos jungia kurias nors viršūnes, o jų svoriai atitinka atitinkamų atkarpų ilgius. Tada algoritmu A* rasti trumpiausią kelią nuo vienos pasirinktos viršūnės iki kitos. Palyginti rezultatus su Deikstros algoritmo rezultatais.

2. Užduoties analizė

A* algoritmas yra labai panašus į Deikstros kelio paieškos algoritmą. Pagrindinis skirtumas – naudojamas papildomas viršūnės įvertis. Paprasčiausias pavyzdys būtų euklidinis atstumas tarp viršūnės ir pabaigos viršūnės. Su šiuo papildomu įverčiu panaikinamas vienas iš Deikstros algoritmo trūkumų – Deikstros algoritmas keliauja per mažiausią svorį turinčius kelius, tad jei briaunos su mažais svoriais eina į priešingą pusę nei ieškoma viršūnė, įvykdoma daug nereikalingų operacijų ir lėtėja algoritmas. Pridėjus atstumo iki ieškomos viršūnės įvertį ir atsižvelgus į jį, to bus išvengta ir gausime A* algoritmą.

Uždavinys. Duotas jungus grafas *G = (V,U)*, rasti trumpiausią kelią nuo viršūnės v_s iki viršūnės v_f. Palyginti rezultatus su Deikstros algoritmu gautais rezultatais.

Metodo idėja. Kad būtų lengviau matyti skirtumus tarp A* ir Deikstros algoritmų, nagrinėkime tokį grafą:



Pav. 1

Tarkime, kad viena briauna prilygsta vienam ilgio matmeniui. Tuomet:

1. Susikuriame masyvus prec (iš kurios viršūnės atėjome), d (kelio svoris) ir L(atstumas iki pabaigos viršūnės);
2. Apskaičiuojame viršūnių ilgius iki pabaigos viršūnės;
3. Susumuojame visus svorius ir pridedame vienetą, kad galėtume inicializuoti masyvą d.

Taip atrodo minėtieji masyvai:

	1	2	3	4	5	6	7
prec	0	0	0	0	5	0	0
d	16	16	16	16	0	16	16
L	6	5	4	3	2	1	0

Pradedame nuo 5 viršūnės. Apžiūrime jos aplinką, perskaičiuojame gretimų viršūnių atstumus, pažymime, kad išanalizavome 5 viršūnę.

	1	2	3	4	5	6	7
prec	0	0	0	5	5	5	0
d	16	16	16	2	0	6	16
L	6	5	4	3	2	1	0

Toliau analizuojame mažiausią d ir L sumą turinčią viršūnę (nepaisant jau pažymėtų) – 4. Atliekame analogiškus veiksmus.

	1	2	3	4	5	6	7
prec	0	0	4	5	5	5	0
d	16	16	4	2	0	6	16
L	6	5	4	3	2	1	0

Sekančioje iteracijoje vėl ieškome mažiausios d ir L sumos. Analizuojame 6 viršūnę, atnaujiname 7 viršūnės svorį – jis tampa 7. Pažymime, kad išnagrinėjome 6 virš. Sekanti mažiausią sumą turinti viršūnė yra 7, išanalizuojame jos aplinka (šiuo atveju nėra). Pažymime, kad išnagrinėjome.

Algoritmas baigia darbą, kai pabaigos viršūnė pažymima išnagrinėta. Trumpiausias kelias randamas einant prec masyvu.

Deikstros algoritmas tuo tarpu būtų aplankęs visas viršūnes, nes jis prioretizuoja mažą svorį turinčias briaunas.

3. Programos algoritmo aprašymas

Programa gaus grafą, aprašytą briaunų matrica, ir grąžins briaunas, nurodančias trumpiausią kelią.

Iš pradžių prec masyvas užpildytas nuliais, d masyvas užpildomas briaunų svoriu + 1, L masyvas užpildomas euklidiniu atstumu nuo kiekvienos viršūnės iki pabaigos viršūnės.

Pradedame nuo pradžios viršūnės. prec masyve nurodome jos vietą, d masyve įrašome nulį. Toliau kiekviename žingsnyje analizuojame dabartinės viršūnės aplinką, surašome briaunų svorius į d masyvą, pažymime, kad išanalizavome dabartinę viršūnę. Tuomet ieškome trumpiausios sumos d ir L atitinkamuose elementuose ir analizuojame mažiausią bendrą svorį turinčią viršūnę. Tai kartojame, kol pabaigos viršūnę bus pažymėta kaip išanalizuota.

Atspausdiname trumpiausią kelią einant per prec masyvą. v_f indekso vietoje žiūrime, iš kurios viršūnės atėjome, toliau žiūrime tos viršūnės indeksą ir t. t. kol pasiekiame pradžios viršūnę.

Pavyzdžiui, su 1 pav. pateiktu grafu bus surastas trumpiausias kelias 5 6 7.

4. Programos tekstas

pagrindinis.m

```
clc; close all; clear all

% Grafo virsuniu koordinates nulinamos, pagal nutylejima virsunes bus isdestomos
ratu.
Vkor = [];

% Testas nr. 1 (duotas)
kelioPradzia = 3;
kelioPabaiga = 8;
V = [1 2 3 4 5 6 7 8];
U = {[1 2 1],[2 3 2],[3 4 4],[3 5 1],[2 5 5],[5 4 1],[4 7 1],[1 6 2],[7 8 1],[5 6
3],[5 8 5], [1 8 15]};

% Testas nr. 2 (tiesi linija)
% kelioPradzia = 5;
% kelioPabaiga = 7;
% V = [1 2 3 4 5 6 7];
% U = {[1 2 3],[2 3 1],[3 4 2],[4 5 2],[5 6 6],[6 7 1]};
% step = 2/(length(V) - 1);
% for i = 1:length(V)
%     Vkor(i,:) = [-1 - step + step * i, 0];
% end

% Testas nr. 3 (grid)
% kelioPradzia = 1;
% kelioPabaiga = 14;
```

```
% V = [1 2 3 4 5 6 7 8 9 10 11 12 13 14];
% U = {[1 2 1],[2 3 1],[3 4 1],[5 6 1],[6 7 1],[8 9 1],[9 10 1],[11 12 1],...
%      [12 13 1],[2 5 1],[3 6 1],[4 7 1],[5 8 1],[6 9 1],[7 10 1],...
%      [8 11 1],[9 12 1],[10 13 1],[11 14 1]};
% step = 0;
% Vkor(1,:) = [-1,0.8];
% for i = 1:3:10
%     Vkor(i+1,:) = [-0.5, 0.8 - step];
%     Vkor(i+2,:) = [0, 0.8 - step];
%     Vkor(i+3,:) = [0.5, 0.8 - step];
%     step = step + 0.4;
% end
% Vkor(14,:) = [-0.5,-1];

disp('Darbo pradzia')

orgraf = 0;  % grafas neorientuotasis
% Pradinio grafo brezimas
arc = 0; poz = 0; Fontsize = 10; lstor = 1; spalva = 'b';
figure(1)
title('Duotasis grafas')
Vkor = plotGraphVU(V,U,orgraf,arc,Vkor,poz,Fontsize,lstor,spalva,kelioPabaiga);
hold on; pause(1)

[d,prec,UU,zingNr,minKelias] = astar(V,U,kelioPradzia,kelioPabaiga,orgraf,Vkor);
% [d,prec,UU,zingNr,minKelias] = deikstra(V,U,kelioPradzia,kelioPabaiga,orgraf,Vkor);

disp( ['Kelio pradzia: ',num2str(kelioPradzia), ' virsune']);
disp( ['Kelio pabaiga: ',num2str(kelioPabaiga), ' virsune']);

disp('Atstumai iki kitu virsuniu  (d masyvas)'); disp(d);
disp('Is kur atejo  (prec masyvas)'); disp(prec);

for i = 1:zingNr
    title(sprintf('Algoritmo kelias:  %d  zingsnis ',i));
    V1 = UU{i};
    U1 = {V1};
    V1kor = [Vkor(V1(1),:); Vkor(V1(2),:)];
    plotGraphVU(V1,U1,0,0,V1kor,0,10,3,'r',kelioPabaiga);
    pause(0.5)
end

for i = 1:length(minKelias)
    title(sprintf('Trumpiausias kelias:  %d  zingsnis ',i));
    V1 = minKelias{i};
    U1 = {V1};
    V1kor = [Vkor(V1(1),:);Vkor(V1(2),:)];
    plotGraphVU(V1,U1,0,0,V1kor,0,10,3,'g',kelioPabaiga);
    pause(0.5)
end

disp('Darbo pabaiga')
```

distance.m

```
function [atstumai] = distance(Vkor, kelioPabaiga)
n = length(Vkor);
distFactor = 2;
atstumai = zeros(1,n);
for i = 1:n
    xDist = abs(Vkor(kelioPabaiga,1) - Vkor(i,1)) * distFactor;
    yDist = abs(Vkor(kelioPabaiga,2) - Vkor(i,2)) * distFactor;
```

```
    atstumai(i) = sqrt(xDist^2 + yDist^2);
end
end
```

astar.m (tai yra koreguotas deikstra.m failas, tačiau pritaikytas, kad atitiktų A* algoritmą)

```
function [d,prec,UU,zingNr,minKelias] =
astar(V,U,kelioPradzia,kelioPabaiga,orgraf,Vkor)
% A* algoritmu apskaiciuoja trumpiausius kelius svoriniame
% grafe nuo virsunes "s" iki likusiu grafo virsuniu.
%
%    Formalûs parametrai
% V     - grafo virsuniu aibe,
% U     - grafo briaunu aibe; [u,v,c]- (u,v) - briauna, c - jos ilgis/svoris,
% s     - pradine kelio virsune,
% orgraf = 0,jei grafas neorientuotasis,
%        = 1,jei grafas orientuotasis,
% Vkor - grafo virsuniu koordinates; parametras nebutinas;
%        jei Vkor nenurodytas arba Vkor =[], tai grafo virsunes
%        isdestomos apskritimu; priesingu atveju - pagal nurodytas
%        koordinates.
% d     - atstumai tarp virsuniu
% prec - is kurios virsunes atejo
% UU    - trumpiausio kelio briaunu aibe
% zingNr - kelio zingsniu kiekis

% Paruosiamieji veiksmai
n = numel(V); m = numel(U);
dz = zeros(1,n);  % virsuniu dazymo pozymiu masyvas
d = zeros(1,n); prec = zeros(1,n);

[atstumai] = distance(Vkor, kelioPabaiga);

svoris = 1;
for i = 1:m
    a = U{i};
    svoris = svoris + a(3);
end
d = d + svoris;
d(kelioPradzia) = 0; prec(kelioPradzia) = kelioPradzia; t = true;

% Gretimumo strukturos apskaiciavimas
GAM = UtoGAM(V,U,orgraf);

zingNr = 0; clear UU;
while (~all(dz) == 1) && t
    minSvoris = min(d(dz == 0));
    if minSvoris == svoris
        disp('Grafas G - nejungusis')
        return
    end

    suminiaiSvoriai = d + atstumai;

    ind = find((suminiaiSvoriai == min(suminiaiSvoriai(dz == 0))) & ~dz);
    k = V(ind(1)); v = prec(k);
    dz(k) = 1;    % nudazome virsune "k"

    % Briaunos dazymas (v,k)
    if k ~= v
        zingNr = zingNr + 1;
        V1 = [v,k];
```

```
        UU{zingNr} = V1;
    end

    % Perskaiciuojame masyvu d ir prec elementus
    a = GAM{k};
    [~,nn] = size(a);
    for i = 1:nn
        u = a(1,i);
        if (dz(u) == 0) && (d(u) > d(k) + a(2,i))
            d(u) = d(k) + a(2,i);
            prec(u) = k;
        end
    end  % for

    if dz(kelioPabaiga) == 1
        tempV = kelioPabaiga;
        i = 0;
        while tempV ~= kelioPradzia
            i = i + 1;
            minKelias{i} = [tempV,prec(tempV)];
            tempV = prec(tempV);
        end

        return
    end

end  %while

return
```

deikstra.m (prie duoto failo pridėta trumpiausio kelio sudėjimas į masyvą spausdinimui)

```
function [d,prec,UU,zingNr,minKelias] =
deikstra(V,U,kelioPradzia,kelioPabaiga,orgraf,Vkor);
% DEIKSTRA funkcija apskaiciuoja trumpiausius kelius svoriniame
% grafe nuo virsunes "s" iki likusiu grafo virsuniu.
%
%    Formalûs parametrai
% V    - grafo virsuniu aibe,
% U    - grafo briaunu aibe; [u,v,c]- (u,v) - briauna, c - jos ilgis/svoris,
% s    - pradine kelio virsune,
% orgraf = 0,jei grafas neorientuotasis,
%        = 1,jei grafas orientuotasis,
% Vkor - grafo virsuniu koordinates; parametras nebutinas;
%        jei Vkor nenurodytas arba Vkor =[], tai grafo virsunes
%        isdestomos apskritimu; priesingu atveju - pagal nurodytas
%        koordinates.
% d    - atstumai tarp virsuniu
% prec - is kurios virsunes atejo
% UU   - trumpiausio kelio briaunu aibe
% zingNr - kelio zingsniu kiekis

% Paruosiamieji veiksmai
n = numel(V); m = numel(U);
dz = zeros(1,n);  % virsuniu dazymo pozymiu masyvas
d = zeros(1,n); prec = zeros(1,n);
svoris = 1;
for i = 1:m
    a = U{i};
    svoris = svoris + a(3);
end
d = d + svoris;
```

```
d(kelioPradzia) = 0; prec(kelioPradzia) = kelioPradzia; t = true;

% Gretimumo strukturos apskaiciavimas
GAM = UtoGAM(V,U,orgraf);

zingNr = 0; clear UU;
while (~all(dz) == 1) && t
    minSvoris = min(d(dz == 0));

    if minSvoris == svoris
        disp('Grafas G - nejungusis')
        return
    end

    ind = find((d == minSvoris) & ~dz);
    k = V(ind(1)); v = prec(k);
    dz(k) = 1;     % nudazome virsune "k"

    % Briaunos dazymas (v,k)
    if k ~= v
        zingNr = zingNr + 1;
        V1 = [v,k];
        UU{zingNr} = V1;
    end

    if k == kelioPabaiga
        tempV = kelioPabaiga;
        i = 0;
        while tempV ~= kelioPradzia
            i = i + 1;
            minKelias{i} = [tempV,prec(tempV)];
            tempV = prec(tempV);
        end

        return
    end

    % Perskaiciuojame masyvu d ir prec elementus
    a = GAM{k};
    [~,nn] = size(a);
    for i = 1:nn
        u = a(1,i);
        if (dz(u) == 0) && (d(u) > d(k) + a(2,i))
            d(u) = d(k) + a(2,i);
            prec(u) = k;
        end
    end  % for
end  %while

return
```

plotGraphVU.m (koreguotos eilutės 91 – 112, kad būtų spausdinamas viršūnių atstumas iki kelio pabaigos)

```
% Nespausdinti atstumu iki kelio pabaigos, nes spausdinant trumpiausia
% kelia paduodamos tik 2 koordinates
if length(Vkor) > 2
    [atstumai] = distance(Vkor, kelioPabaiga);
end

% virsuniu braizymas:
```

```
for i = 1:nv      % ciklas per virsunes
    x = Vkor(i,1);  y = Vkor(i,2);       % V(i) virsunes koordinates
    ccc = [0.8 0.9 1];   if poz == 1, ccc = 'r';     end
    rectangle('Position',[x-r,y-r,2*r,2*r],'Curvature',[1,1],'FaceColor',ccc);
    % uzrasomas virsunes numeris:
    if abs(V(i))<10,   str = sprintf('%d',abs(V(i)));   shiftx = 0.2*r;
    elseif abs(V(i))<100,   str = sprintf('%2d',abs(V(i)));   shiftx = 0.4*r;
    else,   str = sprintf('%3d',abs(V(i)));   shiftx = 0.6*r;
    end
    text(x-shiftx,y,str);
    if length(Vkor) > 2
        strDist = sprintf('%.1f',abs(atstumai(i)));
        text(x-shiftx,y+0.13,strDist,'Color','blue');
    end
end
```

5. Testavimo pavyzdžiai

Buvo panaudoti trys testavimo pavyzdžiai:

Pirmas testas

Pirmojo pavyzdžio grafas pavaizduotas 2 pav., kai duota viršūnių matrica:

```
V = [1 2 3 4 5 6 7 8];
```

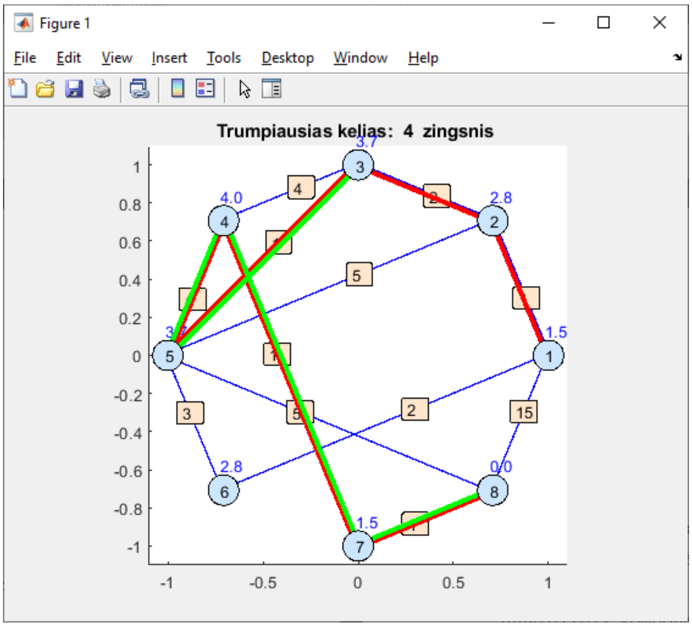
Duota briaunų matrica:

```
U = {[1 2 1],[2 3 2],[3 4 4],[3 5 1],[2 5 5],[5 4 1],[4 7 1],[1 6 2],[7 8 1],[5 6
3],[5 8 5], [1 8 15]};
```

Ir pradžios bei pabaigos viršūnės:

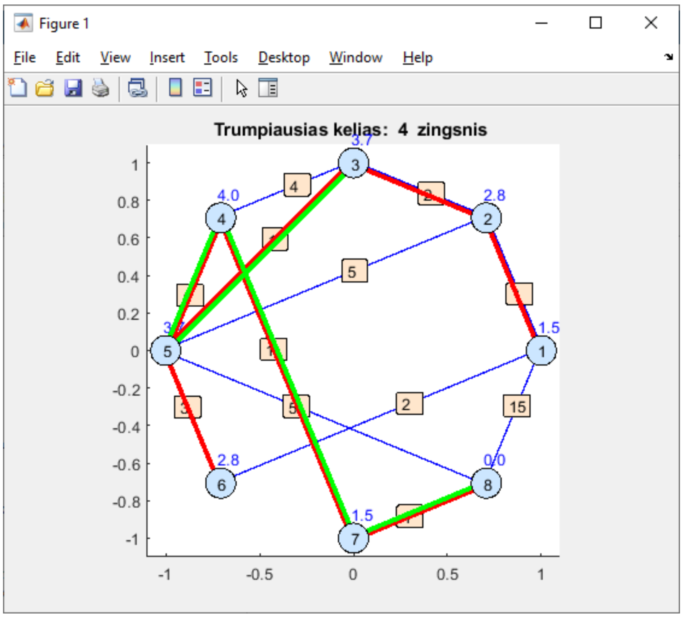
```
kelioPradzia = 3;
kelioPabaiga = 8;
```

Trumpiausias kelias keliaujant viršūnėmis 3, 5, 4, 7, 8. Raudonai pažymėtos algoritmu eitos briaunos, žaliai rastas trumpiausias kelias. Programos rezultatas:



Pav. 2. Pirmo testo trumpiausias kelias

Palyginimui Deikstros algoritmas šį kelią randa aplankęs daugiau viršūnių:



Pav. 3. Trumpiausias kelias rastas pagal Deikstros alg.

Antras testas

Antrojo pavyzdžio grafas pavaizduotas 4 pav., kai duota viršūnių matrica:

```
V = [1 2 3 4 5 6 7];
```

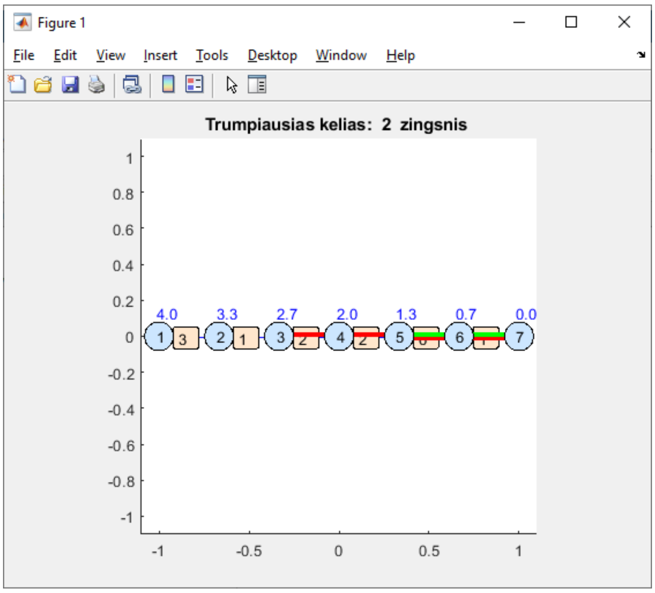
Duota briaunų matrica:

```
U = {[1 2 3],[2 3 1],[3 4 2],[4 5 2],[5 6 6],[6 7 1]};
```

Ir pradžios bei pabaigos viršūnės:

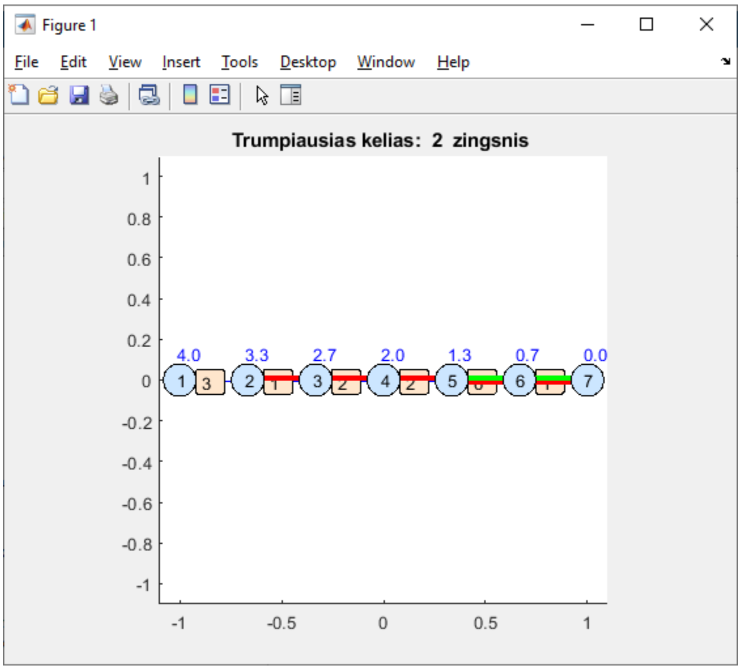
```
kelioPradzia = 5;
kelioPabaiga = 7;
```

Trumpiausias kelias keliaujant viršūnėmis 5, 6, 7. Programos rezultatas:



Pav. 4. Antro testo trumpiausias kelias

Tuo tarpu Deikstros aplanko viena viršūne daugiau, nors ji yra visiškai kitoje pusėje nei pabaigos viršūnė.



Pav. 5. Antro testo kelias rastas pagal Deikstros alg.

Trečias testas

Trečiojo pavyzdžio grafas pavaizduotas 6 pav., kai duota viršūnių matrica:

```
V = [1 2 3 4 5 6 7 8 9 10 11 12 13 14];
```

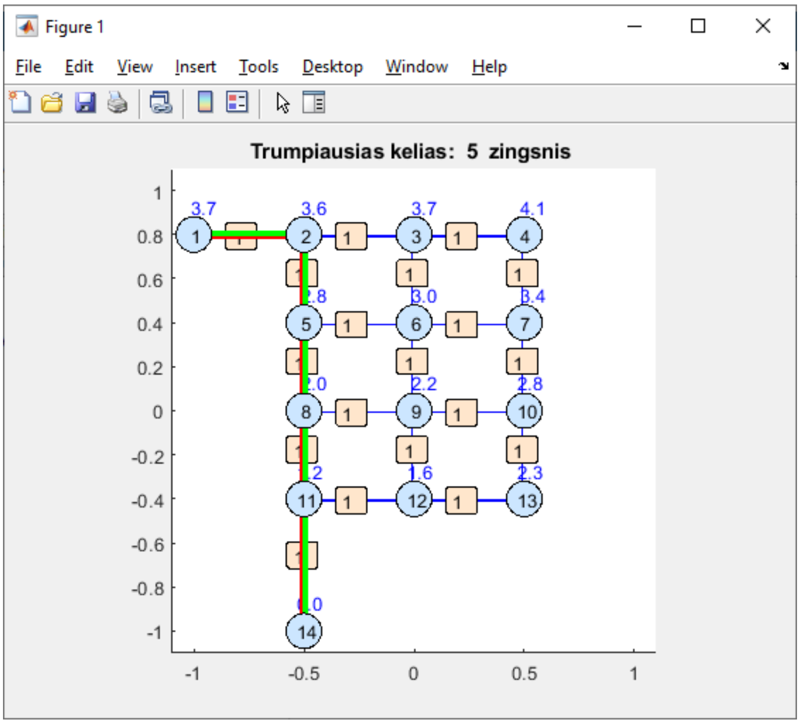
Duota briaunų matrica:

```
U = {[1 2 1],[2 3 1],[3 4 1],[5 6 1],[6 7 1],[8 9 1],[9 10 1],[11 12 1],...
    [12 13 1],[2 5 1],[3 6 1],[4 7 1],[5 8 1],[6 9 1],[7 10 1],[8 11 1],...
    [9 12 1],[10 13 1],[11 14 1]};
```

Ir pradžios bei pabaigos viršūnės:

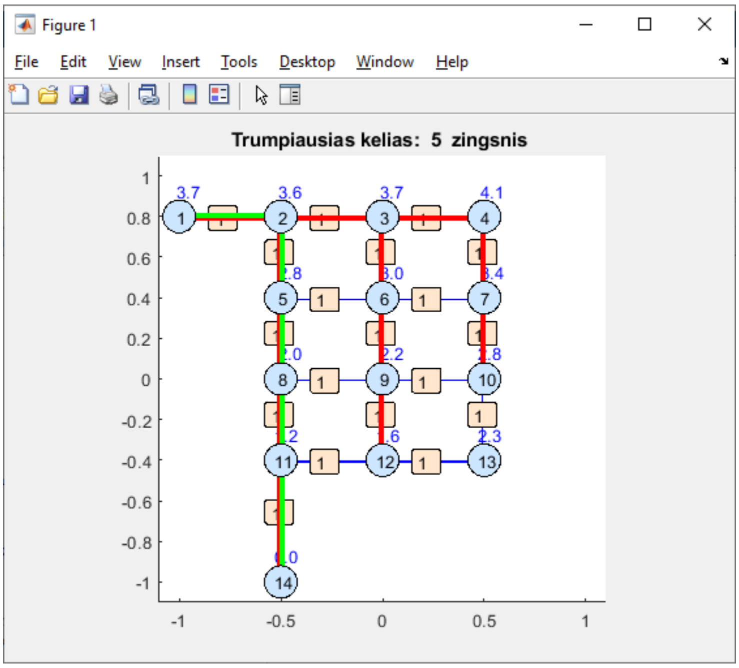
```
kelioPradzia = 1;
kelioPabaiga = 14;
```

Trumpiausias kelias keliaujant viršūnėmis 1, 2, 5, 8, 11, 14. Programos rezultatas:



Pav. 6. Trečio testo trumpiausias kelias pagal A.*

Šiuo atveju, Deikstros algoritmas atlieka paiešką artimai paieškai į plotį.



Pav. 7. Trečio testo rezultatai naudojant Deikstros alg.

6. Išvados

Programa veikia teisingai – randamas trumpiausias kelias tarp pradžios ir pabaigos viršūnių.

Implementuodamas A* algoritmą pastebėjau, kad labai svarbu tinkamai parinkti atstumo dauginimo koeficientą (jo reikėjo, nes grafo viršūnių koordinatės priklauso intervalui [-1; 1]. Todėl apskaičiuoti atstumai yra gana maži). Nustačius per didelį, tikėtina, kad algoritmo grąžintas kelias nebus trumpiausias.

A* pranašumas labiausiai matosi palyginus pav. 6 ir pav. 7. Realiame gyvenime toks grafas atitiktų blokais išdėstytą miesto dalį. A* algoritmas pirmenybę teikia viršūnės, artimesnėms ieškomai viršūnei, todėl, palyginus su Deikstros algoritmu, atlieka mažiau netikslingų veiksmų. Šiame pavyzdyje Deikstros algoritmas nedaug skiriasi nuo paieškos į plotį.

Atstumo apskaičiavimas taip pat daro didelę įtaką algoritmo veikimo greičiui. Pavyzdžiui, apskaičiavus per didelius atstumus, A* beveik nesiskirs nuo Deikstros algoritmo atliekamų veiksmų kiekiu.

7. Literatūros sąrašas

1. Matlab dokumentacija http://www.mathworks.se/help/index.html (žiūrėta 2019-12-08)
2. „A* Search Algorithm“, svetainė „GeeksforGeeks“ https://www.geeksforgeeks.org/a-search-algorithm/ (žiūrėta 2019-11-25)
3. „A* (A Star) Search Algorithm – Computerphile“ https://www.youtube.com/watch?v=ySN5Wnu88nE (žiūrėta 2019-11-25)